ISI/TM-78-10
March 1978
ARPA ORDER NO. 2223

HP/MME Terminal

Application Specification

Don Oestreicher
Paul Raveling
Robert Stotz

INFORMATION SCIENCES INSTITUTE
UNIVERSITY OF SOUTHERN CALIFORNIA
4676 Admiralty Way/ Marina del Rey/ California 90291
(213) 822-1511

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DDC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: ISI/TM-78-10	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): HP/MME Terminal: Application Specification	5. TYPE OF REPORT & PERIOD COVERED: Technical Manual
6. PERFORMING ORG. REPORT NUMBER	
7. AUTHOR(s): Don Oestreicher, Paul Raveling, Rob Stotz	8. CONTRACT OR GRANT NUMBER(s): DAHC 15-72-C-0308 ARPA Order-2223
9. PERFORMING ORGANIZATION NAME AND ADDRESS: USC/Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90291	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: ARPA Order #2223
11. CONTROLLING OFFICE NAME AND ADDRESS: Defense Advanced Research Projects Agency, 1400 Wilson Blvd., Arlington, VA 22209	12. REPORT DATE: March 1978
	13. NUMBER OF PAGES: 43
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): -------	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This document is approved for public release and sale; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

----

18. SUPPLEMENTARY NOTES

----

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

terminals, editing, automatic formatting, user interface, interactive terminals, CRT, man-machine interaction

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This document describes the black box characteristics of the modified HP2649A terminal that ISI has developed for the Military Message Experiment (HP/MME). The communication interface between the terminal and the host computer is described, both in terms of the syntax and the semantics. The communication line discipline (protocol) is also described.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

407 952

JOB

ISI/TM-78-10
March 1978
ARPA ORDER NO. 2223

HP/MME Terminal

Application Specification

Don Oestreicher
Paul Raveling
Robert Stotz

INFORMATION SCIENCES INSTITUTE

UNIVERSITY OF SOUTHERN CALIFORNIA

4676 Admiralty Way/ Marina del Rey/ California 90291
(213) 822-1511

THIS RESEARCH IS SUPPORTED BY THE ADVANCED RESEARCH PROJECTS AGENCY UNDER CONTRACT NO. DAHC15 72 C 0308, ARPA ORDER NO. 2223.

CONTENTS

ABSTRACT

This document describes the black box characteristics of the modified HP2649A terminal ISI has developed for the Military Message Experiment (HP/MME). The communication interface between the terminal and the host computer is described, both in terms of the syntax and semantics. The communication line discipline (protocol) is also described.

I. INTRODUCTION

This document describes the black box characteristics of the modified HP2649A terminal ISI has developed for the Military Message Experiment (HP/MME). The communication interface between the terminal and the host computer is described, in terms of both syntax and semantics. The communication line discipline (protocol) is also described.

The HP/MME terminal consists of a Hewlett-Packard 2649A terminal with minor physical modifications and special firmware developed by ISI. The firmware is contained in programmable read-only memories (PROM), which mount on special boards designed and produced by ISI. These boards plug into the 2649 in place of the HP control memory. Appendix V is a list of the 2649 modules that go into an HP/MME terminal.

Communications between the application program in the host computer and the terminal consist of blocks of data representing a complete request from the application program to the terminal (Dispatch) or a complete report of some condition in the terminal to the application program (Notice).

The basic output model supported by the terminal starts with windows of text (up to 7 may be allocated at any time) which may or may not be mapped onto the screen. These windows have upper and lower margins for storage of text which can be "scrolled" on screen. Each window is divided into domains of not more than 100 characters. Domains have various formatting and editing attributes (e.g., enterable, editable, underlined, etc.).

The basic input model is that the user is editing/modifying domains. After a user has modified the contents of a domain and moves the cursor out of that domain, the new contents are transmitted to the host via a Change Domain Notice. Any changed domains are also transmitted prior to the transmission of a break/function key entered by the user.

The terminal does its own screen formatting, within the constraints assigned to domains by the host. The terminal also handles its own memory management. When it needs to make space available it "scrolls" data out of memory and informs the computer by means of a Scroll Notice.

The terminal is basically a half-duplex device and at any time is either in input state (keyboard active) or output state (computer active). Care has been taken to allow the application program as much flexibility as possible, but the design still dictates that certain dispatches cannot be safely transmitted by the computer during the input state. These dispatches are accepted by the terminal, but anomalous results might later appear to the application program or the user. These dispatches are noted throughout the document. During output state the keyboard is inhibited from activating the terminal, although the keystrokes may be buffered if the user types ahead.

For other reasons, the terminal might discover that it cannot execute a request from the application program. In these situations an Error Notice is generated by the terminal which identifies the dispatch which was ignored by the terminal. Error conditions are noted in sections discussing the dispatches and are listed in Appendix IV.

The terminal has a total of 16K bytes for display data and domain data. Switch K on the Keyboard Interface card in the terminal controls the allocation of this memory. With Switch K off, upon power up or Reset, memory will be allocated as 12K bytes for display data storage and 4K bytes for domain storage, which is equivalent to 256 domains. With Switch K on, memory will be allocated as 8K for display data and 8K for domains (512 domains).

The terminal also contains a low level debug program, which may be entered by resetting the terminal with Switch J on the Keyboard Interface card activated (on). This debugger (Roach) is described in a separate document.

II. DISPATCHES

Each dispatch is represented by a string of 7-bit characters for the purpose of transmission from the application program to the terminal. The first byte is the total number of bytes in the dispatch (including this count byte). The second byte is the function code. Following bytes are the parameters for the function. All parameters are required, of fixed format, and (except for a possible single text parameter) of fixed size.

The rest of this section is organized in 5 subsections.

A. Window Dispatches

These dispatches provide the capabilities for window allocation and mapping.

B. Domain Dispatches

These dispatches provide the capabilities for creation and modification of domains.

C. Cursor Dispatches

These dispatches provide for controlling the position of the cursor.

D. Flash Dispatches

These dispatches provide the capability for unstructured (write-only) output to the screen.

E. Miscellaneous Dispatches

Appendix III gives an example of putting a new window on the screen to demonstrate the use of these dispatches.

A. WINDOW DISPATCHES

The first level of text structuring within the HP/MME is that of windows. There are seven possible windows identified by a window number (1 through 7). Windows may be mapped onto the screen to allow the user to view them. Mapping and unmapping windows already stored in the terminal allow the application program to change the user's screen quickly.

The Window dispatches are:

1) ALLOCATE
2) DEALLOCATE
3) MAP
4) UNMAP
5) CLEAR
6) CHANGE ATTRIBUTES

1. Allocate Window Dispatch

<1> <WINDOW> <ATTRIBUTES> <SECURITY LEVEL>

A Window is allocated, but is not mapped on to the screen. The identifier of the window is given in the <WINDOW> byte. An Error Notice is generated if this window identifier is greater than 7 or if the window is already allocated. The first parameter defines the window attributes, which are shown below:

Window Attribute Bits (octal)

100 - Do not reclaim (i.e., No Scroll Notices) from the Upper Margin.
40 - Do not reclaim (i.e., No Scroll Notices) from the Lower Margin.
20 - Mask for Margin Reclaim Control.
10 - Upper Margin Vacancy Control (1 = Do Not send Vacancy Notices on Upper Margin).
4 - Lower Margin Vacancy Control (1 = Do Not send Vacancy Notices on Lower Margin).
2 - Mask for Lower Margin Vacancy Control.
1 - Size (0 = small, 1 = large).

The discussion of "Change Window Attributes" dispatch (Section II A 6) describes the detailed operation of these attribute bits.

<SECURITY LEVEL> denotes the classification level for the window. Valid security codes are:

Unclassified - "d"
Confidential - "c"
Secret - "S"
Top Secret - "T"

2. Deallocate Window Dispatch

<2> <WINDOW>

Deallocate the specified window, releasing its contents. If the window is mapped onto the screen, the screen area is cleared and made available for mapping.

3. Map Window Dispatch

<3> <WINDOW> <SCREENLINE> <SCREENLINE>

The specified window is mapped onto the visible screen. The two screen line parameters (respectively, inclusively) specify the first and last lines on the terminal that the window is to occupy. If the window is already mapped onto the screen or the lines are already in use, an Error Notice is generated and the Map Dispatch ignored.

4. Unmap Window Dispatch

<4> <WINDOW>

The specified window is removed from the visible screen. The position of the cursor within the window and the display text is retained. If the window is already unmapped, an Error Notice is generated and the dispatch is ignored. This dispatch cannot be safely transmitted during input state, as the user might be editing the window.

5. Clear Window Dispatch

<5> <WINDOW>

Clears the contents of the specified window, leaving the window allocated. If the window was on the screen, that area of the screen is cleared, but still remains mapped.

6. Change Window Attributes

<44> <WINDOW> <ATTRIBUTES> <SECURITY LEVEL>

The window attributes and security level of the specified window (described in Allocate Window Dispatch) are replaced with new attributes and security level. There are three types of attributes. Window Size is always set by this dispatch. Reclaim Controls and Vacancy Controls each have mask bits associated with them, so they may be independently controlled.

The Reclaim Controls (bits 100 and 40 with mask bit 20) allow the application program to "lock" portions of text into display memory. When the terminal surveys its memory

assignment to windows in order to reclaim memory for subsequent use (i.e., generate Scroll Notices), it will examine these bits of the Window Attribute. Any window margins marked "Not Reclaimable" will not be scrolled out of memory. Of course, if the application program locks too much data in memory, the terminal will eventually run out of space (the terminal limit is 256 domains and 12,000 bytes of display data). It will then generate an Error Notice (Error Type 12) and will reject all subsequent dispatches except ones that free up memory (Delete Domain, Clear Window, Deallocate Window or Reset), until free memory exceeds the minimum threshold.

The Not-Reclaimable attribute for windows gives a mechanism to avoid a race condition where the host computer is writing information relative to data that the terminal is simultaneously scrolling out of its memory. To avoid this condition, the host sets the window attributes to "Not Reclaimable" for the window it plans to write into. Now the host has "control" over that window and it is free to write into it without fear of conflict with the terminal. Whenever the terminal generates a Vacancy Notice on the margin of a window (see Section III B 2), it will set the Not-Reclaimable bit for that same margin. This inhibits the terminal from sending Scroll Notices on that margin. When the host is finished sending data, it must set the window margins to "Reclaimable".

The Vacancy Control bits (bits 20 and 4) have similar functions relative to Vacancy Notices. If the application program wishes to receive Vacancy Notices for a margin of a window, it zeroes the appropriate Vacancy Control bit. If it does not want Vacancy Notices (e.g., if there is no further data to send for the margin of that window) it sets the bit.

When the terminal determines that a mapped window margin is below its vacancy threshold (currently set at half the number of lines mapped), it will examine if vacancies are permitted for this window margin. If so, it will send a Vacancy Notice and set the Not-Reclaimable bit for the margin and a global No Vacancy flag. The terminal is now inhibited from sending Scroll Notices from that margin or any further Vacancy Notices on any margin. When the application program receives the Vacancy Notice, it normally responds by sending additional data for the margin. When it has sent an appropriate amount of data, the host sets both margins for the window to Reclaimable. This dispatch (Change Window Attributes with bit 20 = 1 and bits 40 and 100 = 0) also clears the global No Vacancy flag automatically, in essence asking for another Vacancy Notice when the conditions warrant it. Note that if the application program determines that there is no further data to send on one or both of the margins, it should set the Vacancy Control bits appropriately with this same dispatch.

The size attribute is simply an indication of the anticipated size of the window. The actual size is determined by how much data is entered in the window, and it normally increases or decreases through time.

Since the terminal sets the Vacancy Control bits asynchronously from the host, the host does not necessarily know the state of these bits. In order that the host does not

inadvertently reset a Vacancy Control bit with a Change Window Attributes Dispatch which was sent for a different function, a mask bit is provided. The terminal will only change the state of a Vacancy Control bit on a Change Window Attributes Dispatch if the corresponding mask bit is a one.

B. DOMAIN DISPATCHES

Within windows the application communicates to the terminal in terms of domains. Domains represent the atomic unit for manipulation of text on the screen. In addition, domains have attributes which apply to its entire contents. Domains may be created and deleted by the terminal or the application program in the host computer. Terminal-generated domain identifiers have a one as their high order bit. To avoid conflict computer-generated domains must have zero as their high order bit.

The dispatches for domain manipulation affect the content of the screen if they apply to a window which is mapped. It is therefore dangerous to send any of these dispatches to mapped windows during input state (keyboard active).

1) CREATE
2) DELETE
3) APPEND
4) MODIFY
5) CHANGE TEXT
6) SPLIT
7) JOIN

1. Create Domain Dispatch

<6> <WINDOW> <DOMID> <LEFTDOMID> <CAPAB> <FORMAT> <HILITE> <TEXT>

Within the specified window, a new named domain <DOMID> is created, to the right of the domain <LEFTDOMID>, with the initial text contents specified by <TEXT>. Text may not exceed 100 characters. The special values 0 and -1 for <LEFTDOMID> denote that the new domain is to be created at the beginning (0) or end (-1) of the window.

The argument <CAPAB> specifies the set of capabilities for this domain, defined as the following bit positions:

Capability Bits for Domains

100 - Enterable
40 - Editable
20 - HERE markers allowed
10 - EOL's allowed
4 - Reserved
2 - Reserved
1 - Reserved

The <FORMAT> argument provides the computer the following control over the format of the display of the domains.

Format Bits for Domains

1 - Start domain at left margin
2,4 - Reserved
10,20,40 - Select Left Margin
000 - Left margin = Column 1
001 - Left margin = Column 9
010 - Left margin = Column 17
011 - Left margin = Column 25
100 - Left margin = Column 33
101 - Left margin = Column 41
110 - Left margin = Column 49
111 - Left margin = Column 57

Bit 1 causes the domain to begin at the left margin of the next line, regardless of where the previous domain ends. The Left Margin Control bits permit indentation of the left margin on a per domain basis. Note that an indented domain may not start at the left margin (bit 1 = 0), in which case only that part of the domain which spills over onto the next line will be indented.

The <HILITE> argument selects combinations of the following features:

HILITE Bits for Domains

1 - Blink
2 - Inverse Video
4 - Underline
10 - Half Brightness
20,40 - Select Character Set
00 - Normal
01 - Alternate Character Set.
10 - Not used
11 - Non-printing character set.
Useful for non-printing fields,
e.g., passwords.

The text of a domain may not exceed 100 characters. Note that Scroll Notices may be generated as a result of the Create Domain Dispatch.

2. Delete Domain Dispatch

<7> <WINDOW> <DOMID>

The domain specified by <WINDOW> <DOMID> is deleted. If the domain is visible the screen is changed appropriately. No Notices are generated.

3. Append Domain Dispatch

<8> <WINDOW> <DOMID> <TEXT>

The text specified by <TEXT> is appended to the end of the specified domain. Note that Scroll Notices may be generated. If the resulting domain is longer than 100 characters, an Error Notice is generated and the dispatch is ignored.

4. Modify Domain Dispatch

<9> <WINDOW> <DOMID> <CAPAB> <FORMAT> <HILITE>

The attributes for the specified domain (described in Create Domain Dispatch) are replaced with new attributes. If the domain is visible the screen is changed appropriately. Scroll Notices may be generated.

5. Change Text Dispatch

<10> <WINDOW> <DOMID> <TEXT>

The text of the specified domain is replaced by the new text. Text may not exceed 100 characters. If the domain is visible the screen is changed appropriately. Scroll Notices may be generated.

6. Split Domain Dispatch

<11> <WINDOW> <OLDDOMID> <NEWDOMID> <INTEGER1> <BEGINNING/END>

A new domain <NEWDOMID> is created with the characteristics of <OLDDOMID> and containing the <INTEGER1> number of characters of the old domain; the old domain is shortened appropriately. If Beginning (0) is specified by the last parameter, the new domain is created to the left of the old domain, with the first <INTEGER1> characters in it; if End (1) is specified, the new domain is created to the right of the old domain, with the last <INTEGER1> characters in it. This dispatch does not change the screen. Scroll Notices may be generated.

7. Join Domain Dispatch

<12> <WINDOW> <KEEPDOMID> <DELDOMID>

The two domains <KEEPDOMID> <DELDOMID> are merged into one <KEEPDOMID>, whose name and attributes prevail. This dispatch may change the screen. If the domains are not adjacent or if the resulting domain is longer than 100 characters, an Error Notice is generated and the dispatch is ignored. Otherwise, no notices are generated.

C. CURSOR DISPATCHES

There is the potential for an implicit cursor position associated with each window in the terminal. This position is used when the user jumps the actual cursor into the window or when the application requests the actual cursor to be placed in the window. The implicit cursor position is established whenever the user moves the actual cursor into the window or the application program sets it through the dispatches below. A window will have no implicit cursor position if one of these two events has not occurred. None of the dispatches in this section can safely be transmitted during input state to any mapped windows.

There are two degrees of freedom associated with the cursor position. The first is the character within the window under which the cursor resides. The cursor must be under an enterable character. The user changes this part of the cursor position with the Move Cursor keys. The application uses the Character Position Dispatch.

The second degree of freedom is where the cursor is on the screen. The user changes this part of the cursor position with the scroll keys. The application uses the Screen Position Dispatch.

The three position dispatches are:

1) CHARACTER POSITION
2) SCREEN POSITION
3) SHOW CURSOR

1. Character Position Dispatch

<13> <WINDOW> <DOMID> <INTEGER1>

This dispatch places the implicit cursor on the character position specified by the integer within the specified domain in the specified window. The character is a zero-based number (i.e., the first character in the domain is position 0). If character position is larger than the domain length, the cursor is placed at the end of the domain. This dispatch does not change the screen.

2. Screen Position Dispatch

<14> <WINDOW> <INTEGER1>

This dispatch is used to establish on what part of the screen the cursor will appear, by placing the line containing the cursor on the specified screen line (relative to the first line of this window on the screen). The first on-screen line of the window is line zero. The screen is scrolled to place the cursor character position on the proper displayed line. If the line number is too large, the window is scrolled until the cursor is on the last on-screen line of the window. Scroll and Vacant Notices may thus be generated. The terminal will not allow the first line in the window to scroll below the top line of the window on screen. If the window is not mapped, the position is remembered and used to determine which lines to display when the window is mapped.

3. Show Cursor

<15> <WINDOW>

This dispatch moves the actual cursor from some other window to the implicit cursor position in the specified window. An Error Notice is generated if the window is not mapped or there is no implicit cursor position in this window.

D. FLASH WINDOW DISPATCHES

The flash window is an area at the top of the screen which is used to transmit unstructured, non-editable text directly to the screen. This screen area cannot be edited, nor can the user move the cursor into it. Flash window dispatches may be safely transmitted at any time.

The two flash window dispatches are:

1) FLASH SIZE
2) FLASH TEXT

1. Flash Size Dispatch

<16> <INTEGER1>

This dispatch sets the size of the flash window. The initial size (following a reset) is zero. As the screen has 24 lines, the parameter must be between 0 and 24; otherwise an Error Notice is generated.

If the flash window size is being increased, the text currently in the window is unchanged. If the new lines are already mapped to another window, an Error Notice is generated and the dispatch is ignored.

If the flash window size is being decreased, the text in the lines to be deleted is cleared from the screen. The other flash lines are not changed. The deleted lines are now available to be mapped to other windows.

2. Flash Text Dispatch

<17> <SCREENLINE> <TEXT>

This dispatch puts the specified text on the specified line of the flash window. The first line of the flash window is numbered zero. Possible conditions which generate Error Notices are that the specified line is not mapped to the flash window or that the text is longer than 79 characters (i.e., the text does not fit on the line).

F. MISCELLANEOUS DISPATCHES

The four miscellaneous dispatches are:

1) BELL
2) CONTINUE
3) RESET
4) SET SCREEN SECURITY LIGHTS

1. Bell Dispatch

<18>

This dispatch, which can be transmitted at any time, rings the bell in the terminal. Care

should be taken when the bell is rung during the input state, as the terminal uses the bell as an error indicator if the user enters an inappropriate local edit function. Thus, during input state, the bell might be misconstrued by the user.

2. Continue Dispatch

<19> <INTEGER1>

Following a break/function key, the terminal stops echoing the user's keyboard input, sets the global No Vacancy flag (see Section II A 6), and enters output state. The host computer is notified of this condition by the receipt of a Break Notice. When the host computer has completed its output and is ready to accept additional input, it transmits a Continue Dispatch to place the terminal back into input state with Vacancies enabled. The <INTEGER1> argument sets a flag, which will determine whether keystrokes will be buffered or thrown away (in which case the bell will be rung) after the next Break. On initial power up and after a Reset, the terminal will lock its keyboard (keystrokes will not be buffered). The host must send a CONTINUE to change this situation. A zero parameter to CONTINUE will lock the keyboard after the next Break. A non-zero parameter will cause keystrokes to be buffered.

3. Reset Dispatch

<20>

The Reset Dispatch initializes the terminal. This includes deallocating all windows, setting the flash window to zero lines, and setting the terminal domain identifier counter back to 20000 (octal).

4. Set Screen Security Lights

<21> <SECURITY LEVEL>

This dispatch sets the screen level security lights to read as the security level specified. This dispatch may be safely transmitted at any time.

III. NOTICES

Each notice is a string of 7-bit characters. Its purpose is to transmit data from the terminal to the application program. The first byte is the number of bytes in the notice. The second byte is the notice code followed by the notice data. All data is always present, of fixed format, and (except for a possible single text datum) of fixed size.

The notices are presented below in three sections:

A. Input Notices

These notices inform the application of user inputs (edits and function keys).

B. Memory Management Notices

These notices inform the application program that domains have been "scrolled" out of the terminal's memory or that there is a need for new data at the top or bottom of a window. They may be generated in response to any dispatch or user action which creates or changes the size of windows or domains. They may also be generated in response to dispatch or user action which changes the screen (e.g., scrolling).

C. Error Notice

This notice is generated in response to a dispatch that the terminal cannot handle. It is also used as an acknowledgment notice for completion of a Reset dispatch.

A. INPUT NOTICES

The input notices allow the application program to know exactly what the user has typed and the current state of the screen model in the terminal. No input notices are generated following a Break Notice until the terminal is put back into input state by a Continue Dispatch.

The five input notices are:

1) CHANGED DOMAIN
2) EXTRACTION
3) BREAK
4) HERE
5) EOL

1. Changed Domain Notice

<0> <WINDOW> <DOMID> <TEXT>

Whenever the user edits a domain, the new contents of the domain are transmitted to the application before the next Break Notice. If the entire domain was deleted, then the length of the text for the contents is zero. These notices are generated irregularly during the input state. The application may not safely respond to these notices via domain modification dispatches (join, split, etc.) until it receives a Break Notice (signifying the end of input state).

2. Extraction Notice

<1> <WINDOW> <OLDDOMID> <NEWDOMID> <BEGINNING/END> <TEXT>

As a user types new data into a domain, at some point the domain exceeds the maximum domain size. When this occurs, the terminal splits off a new domain. The new domain inherits the characteristics of the old, and its contents are sent as <TEXT>. Eventually a Changed Domain Notice is sent for the original old domain. <BEGINNING/END> is 0 when the new domain is to the left of the old domain; it is 1 when the old domain is to the left of the new.

Another type of Extract Notice is generated as an adjunct to Scroll Notices (see Section III 3 B). In this instance a domain which is within the maximum domain size limitations may be split in order to scroll off data at even line boundaries. In this case a Change Domain Notice is not generated for the domain from which the extraction is made. For this type of extraction BEGINNING/END is 2 if the new domain is to the left of the old, and BEGINNING/END is 3 if the old domain is to the left of the new.

3. Break Notice

<2> <WINDOW> <DOMID> <LOC IN DOMAIN> <WINDOW LOC> <BREAK CODE>

Whenever the user hits a break/function key, the application program is notified through a Break Notice. Before the Break Notice is actually sent, any domain change that has not yet been sent to the host will be reported via a Change Domain Notice. The Break Notice includes two pieces of information. First is the position of the cursor at the time the key was hit. This includes the window, the domain within the window, the character within the

domain (zero is the first character) represented as an <INTEGER1>, and the line within the mapped part of the window represented as an <INTEGER1>. The second piece of information is the break code or function key number (<INTEGER1>). Since the Break Notice is initiated by a Break Dispatch, which locks the keyboard and the global No Vacancy flag, the application program must issue a Continue Dispatch when it has completed processing the Break.

There are two additional notices to handle special local edit keys. These keys are not active unless the domain is enabled for them; therefore the application does not need to expect them in all situations.

4. HERE Notice

<3> <WINDOW> <DOMID> <LOC IN DOMAIN> <MDOMID> <ADOMID>

In Sections V and VI the user interface is described. HERE is a key which marks the current position of the cursor and generates a Here Notice. When the user places a Here marker on a character, that character is HILITED to show the user where the marker is located. The marker is also made to be a non-editable domain. In order to do this, the terminal must create domains and inform the application so that it can reflect these changes in its model of what is contained in the terminal.

The first three data in the notice define where the cursor (and now the Here marker) is. The last two parameters depend on where the Here occurs within the original domain.

Case Analysis

There are four cases for a Here marker. They are:

a. single character domain marked

The Here domain identifier <MDOMID> contains the original domain identifier and the auxiliary domain identifier <ADOMID> contains 0. <LOC IN DOMAIN> is 0. No domains are created, the original domain has been HILITED and set non-editable.

b. first character in domain marked

The <MDOMID> is a terminal-created domain identifier containing the first character of the old domain. The old <DOMID> now contains the rest of the original domain. <ADOMID> is 0, as is <LOC IN DOMAIN>.

c. last character in domain marked

The <MDOMID> is a terminal-created domain identifier which now contains the last character of the old domain. The old <DOMID> now contains the beginning of the original domain. <ADOMID> is 0.

d. middle character in domain marked

The <MDOMID> is a terminal-created domain identifier which now contains the marked character. The old <DOMID> now contains the beginning of the original domains, and the <ADOMID> is another terminal-created domain identifier containing the end of the original domain.

5. *EOL Notice*

<4> <WINDOW> <DOMID> <LOC IN DOMAIN> <NDOMID> <SPECIAL>

When the user hits the "Return" key, an EOL Notice is generated (See Section VI D 4). The terminal attempts to handle carriage return in a way that appears natural to the user yet is compatible with the terminal controlling the formatting of the screen. Carriage return is normally meant to indicate that the next line is to begin at the left margin, regardless of where the previous line ends. A carriage return at the beginning of a line inserts a blank line (a formatted domain with a single space), and can be used to indicate the beginning of a new paragraph.

When the user inputs a carriage return, the terminal inserts a space at the cursor position, splits the current domain behind the inserted space, and sets the second domain to have the start-at-left-margin format bit on. Two special cases add to the complexity in this notice, which derives from the desire to have the user be able to create lasting blank lines naturally whether the carriage return is entered at the end of the window (i.e., entering new data) or in the middle of a window (i.e., editing old data) and regardless of the domain structure surrounding the cursor.

The first two arguments specify the window and domain of the cursor position at the time of the carriage return. <LOC IN DOMAIN> is the location in the domain of the inserted space. <NDOMID> is the new domain created. <SPECIAL> is a 2 byte integer to identify 2 special cases. The details of the EOL function and the notices generated are best described with several examples, which are given in Appendix II to this document.

B. MEMORY MANAGEMENT NOTICES

When a window is scrolled, the margin of off-screen text either above or below the screen gets larger, while the other margin diminishes. In order to give apparent continuity to a large document, the terminal attempts to keep reasonable margins on both sides. Therefore at some point, data is forced out of one margin, making memory available for data at the end of the other. Through window attributes the application program can control whether the terminal will try to trim and fill the margins of a window or not.

Since memory is dynamically allocated, any of a number of operations may spawn the need for more memory, causing the terminal to scroll data out of a window. The dispatches that may have this effect are noted. The memory management notices inform the application of this, so that it might provide new text to the terminal.

The two memory management notices are:

1) Scroll
2) Vacant

1. Scroll Notice

<5> <WINDOW> <DOMID> <TOP/BOT>

The terminal decides to remove a number of domains from its memory; it sends this notice to inform the application program that this domain and all the domains above (if <TOP/BOT> is 0) or below (if <TOP/BOT> is 1) are no longer in the terminal. The application must keep track of what is in the terminal so that it can respond properly to Vacancy Notices and return the proper text to the terminal for the user to view.

2. Vacancy Notice

<6> <WINDOW> <TOP/BOT> <FREE LINES>

When more text is needed at one end of a window to resupply the margin, the terminal generates a Vacancy Notice. This notice specifies where more text is needed (<TOP/BOT>) and how many lines are available. When a Vacancy Notice is generated, the terminal also sets a global No Vacancy flag. This flag must be reset by the host (see Section II A 6) before another vacancy can be generated.

C. ERROR NOTICE

<7> <ERROR CODE> <FUNCTION CODE> <BYTE1> <BYTE2> <BYTE3>

Whenever a dispatch cannot be processed an Error Notice is generated. This is a six-byte notice which specifies:

1. A single-byte error code,

2. A single-byte which is the dispatch function code of the dispatch in error, and

3. The first, second and third parameter bytes of the dispatch in error.

Appendix IV lists the error codes and conditions that cause them.

IV. DATA FORMATS

There are three syntactic data formats. These include two types of integers and text strings. This section also discusses three semantic data types which are used throughout the document (<WINDOW>, <DOMID>, and <SECURITY LEVEL>).

A. SYNTACTIC DATA TYPES

1. <INTEGER1>

This is a single byte and can represent the numbers 0-177 (octal). All parameters except text, domain ID's and <SPECIAL> in the Notice are of this type.

2. <INTEGER2>

This is a double byte and can represent the numbers 0- 37777 (octal). All domain ID's and <SPECIAL> in the EOL Notice are of this type.

3. <TEXT>

<TEXT> is represented as ASCII characters. The terminal provides 128 printing characters (including Space). What are normally considered control characters (e.g., Carriage Returns, Bell, Escape) will print as unique symbols. The HP 2645 manual shows the symbol set for these characters. The length of the text is determined from the total dispatch (or notice) length, given as the first byte of the dispatch (or notice). This byte is of type <INTEGER1>.

B. SEMANTIC DATA TYPES

1. <WINDOW>

A window identifier is represented as an <INTEGER1> of value 1 through 7. In all dispatches except Allocate, if the window has not been allocated, an Error Notice is generated for the dispatch referencing the undefined window. In the Allocate case, an Error Notice is generated if the window is defined, or is greater than 7.

2. <DOMID>

A domain identifier is represented by an <INTEGER2>. The application generates identifiers 1 through 17777. The terminal generates identifiers 20000 through 37776. The identifiers 0 and 37777 are reserved for special cases of left domain in the Create

Dispatch. In all dispatches except Create, if the domain has not been created, an Error Notice is generated for the dispatch referencing the undefined domain. In the Create case, an Error Notice is generated if the new domain is already defined or a terminal identifier is supplied.

3. <SECURITY LEVEL>

A security level is represented by an ASCII code. If any other than the four defined codes are received, an Error Notice is generated. The codes have been chosen to be both easy to remember and different in many bits, one from another. The codes are:

Unclassified	-	"d"
Confidential	-	"c"
Secret	-	"S"
Top Secret	-	"T"

V. LOCAL EDITING

In response to user inputs from the terminal keyboard, the terminal software prepares and queues up "internal" dispatches similar to those generated by the application program in the host computer to perform the user requested service. Each key causes a dispatch of some sort. If the user request cannot be fulfilled, or the request was erroneous, the request is ignored and the bell on the terminal rung. The application program may also send any of these dispatches. The terminal does not distinguish their source. As described in Section I the terminal is basically a half-duplex device and at any time is either in input state (keyboard active) or output state (computer active). It is in input state that the user may edit the screen contents. During the output state, when the computer is in control, the keyboard may be disabled or keystrokes may be buffered depending on the argument used in the last Continue Dispatch sent by the application program.

The keys organize into categories: cursor movement, cursor deletions, window control, printing symbols, function keys, and a few special keys. Window control keys are necessary because typically the screen can show only a portion of the data stored in a window. In these cases the additional data is stored in margins, above and below the screen. Window control keys allow the user to move this data onto the screen.

The data contents of windows are modified through the delete keys, printing symbols, and a few of the special keys, as described below. The printing symbols are inserted at the cursor positions and do not "overwrite." The terminal automatically formats the screen as data is inserted. Data wraps from line to line on word boundaries, where words are delineated by space or end of line. On deletion, the screen is not reformatted until the user moves the cursor to a new line.

In general the only keys that always cause a notice to be sent to the application computer are the function keys and two special keys, HERE and Carriage Return. Other keys, such as deletion or printing symbols, do not necessarily cause notices to be sent immediately, although eventually the results of all editing operation are reported.

VI. LOCAL DISPATCHES

Each dispatch is represented by a string of seven-bit characters. The first byte is the total number of bytes in the Notice. The next byte is the function code, followed by the parameters (if any) for the function. All parameters are required and of fixed format.

A. CURSOR MOVE DISPATCHES

The following dispatches control the movement of the cursor within a particular window. The cursor move dispatches only move the cursor and do not change the data on the screen. Error conditions such as attempting to move the cursor off the screen cause the terminal bell to be rung and the request ignored. Non-enterable domains are skipped over.

Function Code	Key	Description
<22>	↑	Move cursor up to previous enterable line
<23>	↓	Move cursor down to next enterable line
<24>	←	Move cursor left one character
<25>	→	Move cursor right one character
<26>	WORD LEFT	Move cursor left one word
<27>	WORD RIGHT	Move cursor right one word
<28>	BACK	Move cursor to beginning of current line; if at beginning, move to end of previous line
<29>	FWD	Move cursor to end of current line; if at end, move to beginning of next line
<38>	UP WINDOW	Move cursor up to the previous window
<39>	DOWN WINDOW	Move cursor down to the next window

Note that the character move requests can move the cursor to the next or previous line to fulfill the request. The terminal constrains the cursor to always be in some enterable domain. Thus a cursor move will occasionally cause the cursor to jump. For example, if the position above the cursor is non-enterable, an UP cursor move will put the cursor in the nearest enterable domain to that position.

If the current line has multiple "fields" on it (defined by non-enterable domains followed by enterable domains) the move cursor back or forward a line will act as "move field." That is, BACK causes the cursor to move to the beginning of the current field or, if at the beginning, to the end of the previous enterable field. FWD causes the cursor to move to the end of the current field or, if at the end, to the beginning of the next enterable field.

B. CURSOR DELETE DISPATCHES

The following internal dispatches delete information on screen. When deletion and subsequent insertion are complete and the cursor has been moved to another line, the window and screen line are compacted. The domain must have the capability "editable" set to allow information to be deleted. If there are multiple "fields" on a line, these operations delete only the adjacent field, not the entire line.

Function Code	Key	Description
<30>	DEL	Delete one character to left of the cursor
<31>	Del	Delete one character at the cursor
<32>	Shifted BACK	Delete the contents of the line to the left of cursor; if at the beginning of the line, the cursor moves to the end of the previous line.
<33>	Shifted FWD	Delete the contents of the line to the right of the cursor; if at the end of the line, the cursor moves to beginning of the next line.
<34>	Shifted WORD LEFT	Delete one word to the left of cursor.
<35>	Shifted WORD RIGHT	Delete one word to the right of cursor.

If the current line has multiple "fields" on it, as described in VIA above, the delete forward or delete back a line acts only on the current field. That is, holding SHIFT down and pushing BACK deletes to the beginning of the current field or, if at the beginning, moves the cursor to the end of the previous enterable field. Holding SHIFT down and pushing FWD deletes to the end of the current field or, if at the end, moves the cursor to the beginning of the next enterable field.

C. WINDOW CONTROL INTERNAL DISPATCHES

These dispatches allow the user to alter which part of the total contents of the window the screen is displaying.

1. Scroll Up Dispatch

   <36>

   Key - ROLL UP
2. Scroll Down Dispatch

   <37>

Key - ROLL DOWN

The text in the screen window is moved one line in the direction specified. Scroll Notices may be generated by this keyboard dispatch (see Section III B).

D. PRINTING SYMBOLS, FUNCTION KEYS, AND SPECIAL INTERNAL DISPATCHES

These dispatches control text insertion, cause Break Notices, and miscellaneous special functions, respectively.

1. Insert Dispatch

<43> <CHARACTER>

Key - Any printing symbol including space, Tab, Control Keys

The character given by <CHARACTER> is inserted at the cursor position. The rest of the text on the line and the cursor move one position to the right. If the domain is not editable, but the adjacent character to the left is in an editable domain, the character is included in that editable domain. If the adjacent character is also not in an editable domain, the bell is rung and the dispatch ignored.

Any printing key hit while the Control key or the Tab key is depressed will store unique printing symbols for the appropriate control code (see HP 2645 owner's manual for the font for these 32 characters). The terminal treats these as any other printing symbol. Thus Control M does not cause the same operations as the Carriage Return key.

2. Break Dispatch

<41> <BREAKCODE>

Key - Any Function Key

First, the terminal keyboard is inhibited and the global No Vacancy flag is set. Then any change to a domain that is not yet reported is reported in a Change Domain Notice. Finally a Break Notice is generated, which reports the particular Function Key struck, through the break code <BREAKCODE>, and the cursor position. The terminal remains in the output state until a Continue dispatch is received from the applications program. During this state user input is queued or ignored (see Section II F 2).

3. Here Dispatch

<40>

Key - HERE

If the current domain does not have the "HERE's allowed" capability, the request is ignored and the terminal bell is rung. If the capability is allowed, the domain at the cursor is split into two or possibly three domains, the character at the cursor position is highlighted as a HERE (by inverting the video level), and a HERE notice is generated (see Section III A 4).

4. EOL Dispatch

<42>

Key - Carriage Return

If the EOL capability for the domain is not enabled, the bell is rung and the request is ignored. If allowed, the domain is split at the cursor position with a space appended to the first part, and the second part made into a new domain with the "start at left margin" bit set.

The appearance on screen is what one would expect a Carriage Return to do, i.e., break the current line and start a new one. An EOL Notice is generated, and Scroll Notices may be generated (see Section III B).

VII. COMMUNICATION PROTOCOL

The HP/MME Terminal communicates with the application host via dedicated full duplex, standard RS232, asynchronous data communication lines. It does not support multidrop or polling. The terminal is designed to operate at up to 2400 baud. The terminal communicates in block mode. Blocks from the terminal are Notices. Blocks to the terminal are Dispatches. The Notices and Dispatches are variable length, but the maximum length of either (e.g., a maximum Create Domain dispatch) is 110 bytes, not counting communication protocol header and trailer.

The terminal requires full duplex communication, although it is basically a two-state device. The keyboard is active in input state, while in output state the key strokes are either ignored (causing a bell) or stored (and have no other effect) until the terminal is returned to input state, at which time they are processed. During input state, the application computer should not send dispatches that may alter mapped windows, but other dispatches may be sent. During output state the terminal will not issue keyboard-generated notices, but Scroll and Error Notices may occur at any time.

The intent of the Communication Protocol design is to provide a minimum reduction in bandwidth.

For error detection, a one-byte block checksum is used. Blocks received in error are retransmitted. Each block is assigned a one-byte sequence number which is used to identify transmissions.

A. TRANSMISSION FORM

```
**********************************
* STX * SN * DISPATCH * ETX * CC *
**********************************
```

Each dispatch or notice is surrounded with header and trailer information. The unique character STX is first sent to identify the beginning of a transmission. The Start of Text byte (STX) identifies the transmission to be a data message rather than one of the control messages, which are described later.

Following the STX is a single-character Sequence Number (SN). This identifies a particular transmission and its sequential position relative to other transmissions. If retransmitted, it carries the same Sequence Number. Sequence numbers are in the range 100-177 octal.

After the sequence number is the body of the Dispatch or Notice. The protocol uses ASCII

control codes. To avoid confusing data in the body of the dispatch (or notice) with these control codes, all dispatch data characters are first filtered and transformed as follows:

1. All data bytes transmitted are in the range 40-177 octal.

2. The data characters with codes 0-37 and 077 octal are transmitted as a two-character sequence:

   a. the character ? (code 077 octal),

   b. the desired character code plus 40 octal (except ? which is not changed).

At the reception side, such characters are transformed back to their original code.

The first data byte of the body is the length of the dispatch (notice). This byte may be any character between 3 (the minimum dispatch length) and 127 octal. Thus it may be sent as a one- or two-character sequence.

The end of the body is identified by another unique character, ETX. The last character of the transmission is a checksum character (CC). This character is used for error detection. It is a seven-bit character generated by the transmitting process as the two's complement of the sum (without end-around carry) of all characters of the original dispatch from (and including) the STX up to (and including) the last ETX. Note that the checksum is on the transmission before control characters are converted to the two-character sequence. Since the CC can be any number, it is also transformed into a two-character sequence if its value is less than 40 octal. The reception process must keep a running sum of received characters until it recognizes the ETX, then add its results to the checksum character. If the result is not zero, the transmission is considered to be in error.

B. CONTROL MESSAGES

Two control messages are used to acknowledge receipt of a transmission. The negative acknowledgment sequence

```
***************
* NAK *  SN  *
***************
```

is sent when a transmission is received in error (either a protocol or checksum error). The SN of this message is the SN of the last good transmission that has been received. Note that the device that generates this NAK is referring to the SN of its received messages, not its transmitted message SN. The other sequence which is used is a positive acknowledgment sequence.

```
*************
* ACK * SN *
*************
```

is sent to indicate successful receipt of a dispatch or notice. The SN of this control message is the highest SN received successfully (i.e., the transmission just received).

C. PROTOCOL

To provide flow control, acknowledgments may be sent after every successful dispatch or notice receipt. If a message is received in error, NAK is always sent, requesting retransmission. This implies the computer and terminal must keep old messages around for some period of time before discarding them. The amount of storage required is a function of the channel delay time and whether the transmission process hangs up until the last transmission is acknowledged. The MME terminal will not send another notice until the previous one has been acknowledged. The computer should not be so restricted. In order to reduce the load on the host processing acknowledgments, the terminal acknowledges only dispatches with sequence numbers 0 or 32 and the last dispatch of a sequence of dispatches. This latter condition is established when the terminal has received no further dispatches for a period of about 2 seconds.

When an error is detected (by the terminal) a NAK message with the last good SN is returned (e.g., SN n-1). Subsequently other messages may be received (e.g., SN n+1, n+2). These will be thrown away and negatively acknowledged. Upon receipt of the NAK identifying the last good message (n-1) the transmitting process should stop any further transmission, resynchronize (see below), and retransmit the message n. If it never sent n, the transmitter should resynchronize. The transmitting process can then return to transmitting subsequent (n+1, ...) transmissions normally.

D. RESYNCHRONIZATION

Since new transmissions are held up whenever a NAK is received, and a subsequent ACK may get garbled and lost, it is important that a transmitting process not wait forever for an ACK. It should pull itself out of wait state after an appropriate delay (5 - 10 seconds) and try to find the state of the receiving process. This is done by the resynchronizing control sequence. This sequence is also used when a NAK or an out-of-sequence ACK is received.

```
*************
* ENQ * ENQ *
*************
```

Upon recognition of this enquire sequence the receiving process should reset its state ready to accept a new STX and send back the control sequence:

```
**************************
* SYN * SN * SYN * SN *
**************************
```

where the sequence number (SN) is the last good dispatch (notice) received. This will identify to the transmitting process what is the proper next transmission. This control sequence is a four-character sequence to minimize the probability of spurious synchronize acknowledgments.

If the transmitting process does not receive an appropriate acknowledgment sequence, it should try a new enquire sequence. If the enquire sequence continues to fail, the transmitting process should attempt to initialize, as described in the next section.

F. INITIALIZATION AND RESET

At start up or when processes get totally confused so they can not resynchronize, both sender and receiver must reset their sequence numbers to some initial state. To do this the following Reset sequence control is provided.

```
************************
* SYN * 0 * SYN * 0 *
************************
```

If the computer sends this sequence to the terminal, the terminal will set its Reception Sequence Number to 0, and will return a special Reset Acknowledgment:

```
************************
* SYN * 1 * SYN * 1 *
************************
```

If the terminal and computer reach a point where they must reset their sequence numbers, they most likely will also have to reset the entire terminal state.

F. CODES

The following ASCII codes are specially recognized by this protocol:

	Octal	Hex	Meaning
STX	002	02	Start of Text
ACK	006	06	Acknowledge
NAK	025	15	Negative Acknowledge
ENQ	005	05	Enquire
ETX	003	03	End of Text
SYN	026	16	Synchronize

CC is used in this document to represent the checksum character, which may be any 7-bit code.

SN is used to represent a Sequence Number Character, which may be any code in the range 100 to 177 (octal).

APPENDIX I
INDEX OF DISPATCHES AND NOTICES

The following table lists each dispatch by name followed by which section contains the description of the dispatch, its function code and its parameters.

DISPATCH	SECTION	CODE	PARAMETERS
Allocate	II A1	1	<WINDOW><ATTRIBUTES><SECURITY LEVEL>
Deallocate	II A2	2	<WINDOW>
Map Window	II A3	3	<WINDOW><SCREENLINE><SCREENLINE>
Unmap Window	II A4	4	<WINDOW>
Clear Window	II A5	5	<WINDOW>
Change Window Attrib.	II A6	44	<WINDOW><ATTRIBUTES><SECURITY LEVEL>
Create Domain	II B1	6	<WINDOW><DOMID><LEFTDOMID><CAPAB><FORMAT><HILITE><TEXT>
Delete Domain	II B2	7	<WINDOW><DOMID>
Append	II B3	8	<WINDOW><DOMID><TEXT>
Modify Domain	II B4	9	<WINDOW><DOMID><CAPAB><FORMAT><HILITE>
Change Text	II B5	10	<WINDOW><DOMID><TEXT>
Split Domain	II B6	11	<WINDOW><OLDDOMID><NEWDOMID><INTEGER1><BEGINNING/END>
Join Domains	II B7	12	<WINDOW><KEEPDOMID><DELDOMID>
Character Position	II C1	13	<WINDOW><DOMID><INTEGER1>
Screen Position	II C2	14	<WINDOW><INTEGER1>
Show Cursor	II C3	15	<WINDOW>
Flash Size	II D1	16	<INTEGER1>
Flash Text	II D2	17	<SCREENLINE><TEXT>
Bell	II E11	18	
Continue	II E2	19	<INTEGER1>
Reset Terminal	II E3	20	
Set Screen Security Lights	II E4	21	<SECURITY LEVEL>
Cursor Up	VI A	22	
Cursor Down	VI A	23	
Cursor Left	VI A	24	
Cursor Right	VI A	25	
Word Left	VI A	26	
Word Right	VI A	27	
Back	VI A	28	
Forward	VI A	29	

Up Window	VI A	38	
Down Window	VI A	39	
Delete Char Left	VI B	30	
Delete At Cursor	VI B	31	
Delete Back	VI B	32	
Delete Forward	VI B	33	
Delete Word Left	VI B	34	
Delete Word Right	VI B	35	
Scroll Up	VI C1	36	
Scroll Down	VI C2	37	
Insert Character	VI D1	43	<CHARACTER>
Break	VI D2	41	<BREAK CODE>
Here	VI D3	40	
EOL	VI D4	42	

The following table lists each notice by name followed by which section contains the description of the notice, its function code number, and its parameters.

NOTICE	SECTION	CODE	PARAMETERS
Changed Domain	III A1	0	<WINDOW><DOMID><TEXT>
Extraction	III A2	1	<WINDOW><OLDDOMID><NEWDOMID><BEGINNING/END><TEXT>
Break	III A3	2	<WINDOW><DOMID><LOC IN DOMAIN><WINDOW LOC><BREAK CODE>
Here	III A4	3	<WINDOW><DOMID><LOC IN DOMAIN><MDOMID><ADOMID>
EOL	III A5	4	<WINDOW><DOMID><LOC IN DOMAIN><NDOMID><SPECIAL>
Scroll	III B1	5	<WINDOW><DOMID><TOP/BOT>
Vacant	III B2	6	<WINDOW><TOP/BOT><FREE LINES>
Error	III C	7	<ERROR CODE><FUNCTION CODE><BYTE1><BYTE2><BYTE3>

APPENDIX H
EXAMPLES OF EOL NOTICES

The table below shows examples of how the terminal handles EOL under different conditions. The lines with []'s show what is on the screen. The format is [(<format code><domid>) text in domain]. The format codes are:

- ? unknown
- + start-at-left-margin format domain
- - non-formatted domain

The lines such as CR@D record user inputs such as carriage return at the D. The notice lines are DM for <DOMID>, LD for <LOC IN DOMID>, and S for <SPECIAL>. Domain identifiers greater than 40 are terminal- created identifiers.

Normal Operation: where user inputs two carriage returns to generate blank line

	Example 1	Example 2	Example 3
Original Screen Model:			
	[(?1)ABC][(?2)DEF]	[(?1)ABCDEF]	[(?1)ABCDEF]
User Inputs:			
	CR@D	CR@D	CR@D
Result:			
	NOTICE:	NOTICE:	NOTICE:
	DM=2,LD=0,S=0	DM=1,LD=3,S=0	DM=1,LD=3,S=0
	[(?1)ABC][(?2)space]	[(?1)ABCspace]	[(?1)ABCspace]
	[(+41)DEF]	[(+41)DEF]	[(+41)DEF]
User Inputs:			
	CR@D	CR@D	CR@space
Result:			
	NOTICE:	NOTICE:	NOTICE:
	DM=41,LD=0,S=0	DM=41,LD=0,S=0	DM=1,LD=3,S=0
	[(?1)ABC][(?2)space]	[(?1)ABCspace]	[(?1)ABCspace]
	[(+41)space]	[(+41)space]	[(+42)space]
	[(+42)DEF]	[(+42)DEF]	[(+41)DEF]

Special Case: where user inputs only one carriage return to generate blank line

	Example 4	Example 5	Example 6
Original Screen Model:			
	[(?1)ABC	[(?1)ABC]	[(?1)ABC]
	DEF]	[(-2)DEF]	[(+2)DEF]
User Inputs:			
	CR@D	CR@D	CR@D
Result:			
	NOTICE:	NOTICE:	NOTICE:
	DM=1,LD=3,S=42	DM=2,LD=0,S=1	DM=2,LD=0,S=0
	[(?1)ABC	[(?1)ABC]	[(?1)ABC]
	space]	[(-2)space]	[(+2)space)]
	[(+41)DEF]	[(+41)DEF]	[(+41)DEF]
	SPECIAL 1	SPECIAL 2	NO SPECIAL
	[(?1)ABC]	[(?1)ABC]	
	[(+42)space]	[(+2)space]	
	[(+41)DEF]	[(+41)DEF]	

Note that in Example 4, the space generated by the EOL is made into a separate domain. The <SPECIAL> argument being a terminal- generated domain ID identifies this special case. In Example 5, the space generated by the EOL is already a separate domain, but it has been converted to a "formatted" domain. This is distinguished by the fact that<SPECIAL> is the value 1. In all other cases <SPECIAL> is 0.

APPENDIX III
INITIAL WINDOW EXAMPLE

Values shown in this example are in octal. A window is normally initialized in four steps:

1) ALLOCATION
2) DOMAIN CREATION
3) CURSOR POSITION
4) DISPLAY

1. Window Allocation

<ALLOCATE> <7> <1> <"S">

This dispatch allocates window 7 as a large window with the security classification Secret. For this example Scroll Notices are not inhibited.

2. Domain Creation

<CREATE> <7> <0,1> <177,177> <170> <0> <0> <TEXT>

This dispatch creates domain 1 at the end of window 7. The domain is enterable and editable with both "Here"s and Carriage Returns allowed. It has no special formatting, hilites, character set or margin. Succeeding domains are created in a similar manner.

3. Cursor Position

<CHARACTER POSITION> <7> <0,4> <0>

This dispatch positions the implicit cursor in window 7 on the first character of domain 4.

<SCREEN POSITION> <7> <1>

This dispatch positions the cursor in window 7 on the second line of this window on the screen.

4. Display

<MAP> <7> <16> <23>

This dispatch maps the window onto lines 16 through 23 (the bottom 8 lines of the

screen). The implicit cursor for this window at the first character of domain 4, is displayed on screen line 17.

If it was required to have the actual cursor moved into the new window a Show Cursor could now be transmitted.

APPENDIX IV
ERROR TYPES

Error Code in Octal	Description
1	Window not allocated
2	Zero Domain Identifier not allowed
3	No such Domain Identifier
4	Flash Size out of range
5	Flash Line over 79 characters
6	
7	(Not Error) Reset Acknowledge
10	No such dispatch code
11	Domain Identifier already exists
12	Window not defined
13	Window already allocated
14	Terminal Memory Full - Must delete some data
15	Domain already mapped
16	Line already mapped
17	Line out of range
20	Domain Length zero not allowed
21	Window either not mapped or not allocated
22	Adjacent Domain Identifier does not exist
23	No such Security Code
24	Internal consistency error on Block Count
25	Internal consistency error on Line Count
26	Cannot put cursor into non-enterable domain
27	No implicit cursor in window
30	Split length not compatible with domain length
31	Domain length too long
32	Not adjacent domains

APPENDIX V
HP 2640 SERIES LOGIC BOARDS
USED IN THE MME TERMINAL

Board Type	Description
HP - 60123	Keyboard I/F
HP - 60086	Asynch Data Comm
HP - 60124	DMA
HP - 60112	Display Control
HP - 60024	Display Enhancement
HP - 60088	Display Timing
HP - 60093	Processor (8080)
ISI - 06B	PROM Memory (Address 0-16K)
ISI - 06B	PROM Memory (Address 16-32K)
HP - 60065	4K Memory (Start Address 44K)
HP - 60101	8K Bottom Plane Memory (Start Address 48K)
HP - 60101	8K Bottom Plane Memory (Start Address 56K)